Description

You are the JavaScript engineer on a project that is developing a new web application that applies a variety of filter effects to a user's uploaded image. You have demonstrated that intensity transformation filters involving only pixel point operations (e.g. grayscale, brighten, sepia, …) have run times that are well within design specifications. Unfortunately, more complex filters involving the processing of a pixel region (e.g. median, sharpen, blur, …) take too long to complete and lock up the browser window in the process. To improve the experience of users applying these filters you have decided to split the image horizontally into roughly equal strips and divide the work amongst multiple Web Workers. The goal of this project is to demonstrate the viability of this idea using the median filter, which is frequently the most computationally intensive of all pixel region processing filters.

Requirements

1. Select a large image to use to test your program (>= 1,000,000 pixels, i.e. ~1000x1000).
2. Create an HTML file that loads your image, resizes a `<canvas>` element, and draws the image on the canvas.
3. Include an `<input type="range" …>` element on your page that allows the user to select 1 to 8 Workers to process the image.
4. Include a `<button>` element on your page that creates and sets up the selected number of Worker objects when clicked. Each Worker must load your processing script. Manage all Worker objects in an Array.
5. For each Worker created:
   - Get an ImageData object from the canvas 2D context that holds a separate horizontal strip of pixel data to be processed. Use the 2D context's `getImageData(…)` method to get each ImageData object.
   - Create a message object containing this ImageData object as well as the (x, y) coordinates of the original canvas where the image data should be redrawn after it is processed.
   - Post a message to the new Worker with this three-part object as its message data.
6. Each Worker object's "message" event should be handled by a function that does the following.
   - Receive processed data from each Worker as a modified ImageData object
   - Use this processed ImageData and the returned (x, y) coordinates to redraw the processed image data back to the canvas at the proper location.
   - Terminate the Worker.
7. Write a separate script file to be loaded and run by your Worker objects.
8. Your Worker script must handle the "message" event (self.`onmessage`…) so that image data is processed and returned.
9. Each Worker script should extract the received ImageData's Uint8ClampedArray object.
   - Note that for the median filter you will need to create a second/new Uint8ClampedArray of the appropriate size to hold your processed pixels.
   - This new Uint8ClampedArray should have dimensions that are <u>smaller</u> than the provided ImageData's Uint8ClampedArray by two rows and two columns. (See explanation below.)
10. When a Worker's processing is complete, replace the ImageData object's Uint8ClampedArray (the `.data` property) to the new Uint8ClampedArray that holds processed pixel data. Use the Uint8ClampedArray object's `.set(…)` method to replace the data. (i.e. `myImageData.data.set( myNewArray );`)
11. You must have at least one main HTML file and one JavaScript file that is used by your Worker objects.
12. You MUST enter header comments into your JavaScript file including (1) File name, (2) Your name, (3) Description and or purpose of the assignment.
13. You MUST comment your code, explaining what you did in each section.
14. Submit your HTML and/or JavaScript files using Canvas under the appropriate assignment.

Hints

To simplify your code, you should exclude from processing the outer border of pixels in the original image. Start processing at row=1, column=1 and end at height=num_rows-1, width=num_columns-1. In spite of this exclusion, note that the median filter makes use of the pixels in these edges. Therefore, the data for pixels at the edges must nevertheless be included in the ImageData posted to each Worker. But, updated pixel data for these pixels will not be calculated.

To split each image into horizontal strips, calculate the height of each strip of pixels using a formula like the one below. Note that the height of the image is reduced by 2 in order to account for the fact that the top and bottom rows of pixels are excluded from processing.

```
var strip_height = Math.ceil( (image_height-2) / num_workers );
```

When generating the ImageData object that holds the strip of pixel data from the canvas to send to each Worker, make sure to include an extra row/column of pixels on all edges. For example, if Worker 1 will process pixels from row 1 through row 99 and column 1 through 99, the Worker should receive an ImageData object holding pixel data from row 0 through row 100 and column 0 through 100. The extra pixel data is required by the median processing filter. Likewise, if Worker 2 will process pixels from row 100 through row 199 and column 1 through 99, it should receive pixel data from row 99 through row 200 and column 0 through 100.

Be careful with the last strip of pixel data. Processing must stop at the second-to-last row of pixels. For example, if the image is 1000 pixels high, the last row to process must be row 999 but the last row to send to a Worker for processing must be row 1000. Because image heights will likely not divide evenly into the number of Workers, adjust your parameters as necessary when calculating the dimensions of the last strip of pixels to process.

In addition to the ImageData object holding a strip of pixel data, the parameters in the message object posted to a Worker should also include the (x, y) coordinates indicating where the processed ImageData object will be redrawn on the original image. When the main script receives a message from a Worker after processing is complete, the main script needs to know where to draw processed pixels and one way to solve that problem is to pass it to the Worker initially so that the Worker can pass it back as part of its response message data. The object you pass to a Worker can be formatted like the following.

```
{ imdata: myImageData, x: x, y: y }
```

Here, (x, y) are the coordinates for where the processed pixel data will be redrawn on the original image. The Worker does not use the x and y parameters to do its processing. It only passes them back to the original script as message data when processing is complete.

Remember that an ImageData object contains a `.data` property holding a Uint8ClmpedArray object with pixel data, as well as `.width` and `.height` properties that indicate the dimensions of the ImageData object. You should use these properties to determine the bounds of the pixels to be processed by the Worker. To avoid processing edge pixels, start at row=1, column=1, and end at width-1, height-1.

When the main script receives a message in reponse from the Worker, after putting the processed ImageData on the canvas, make sure to terminate the Worker using its `.terminate()` method.